INTRODUCTION TO COMPUTER SCIENCE · WEEK 9 · STATION III

Self-Reference II

Will this program ever stop? One machine that can be any machine. And every interesting question about meaning.

IN THE TERMINAL

Does it stop?

```
// loop.c
uint64_t c;
uint64_t main() {
c = 1;
while (c != 0)
c = c + 2;
return c;
}
```

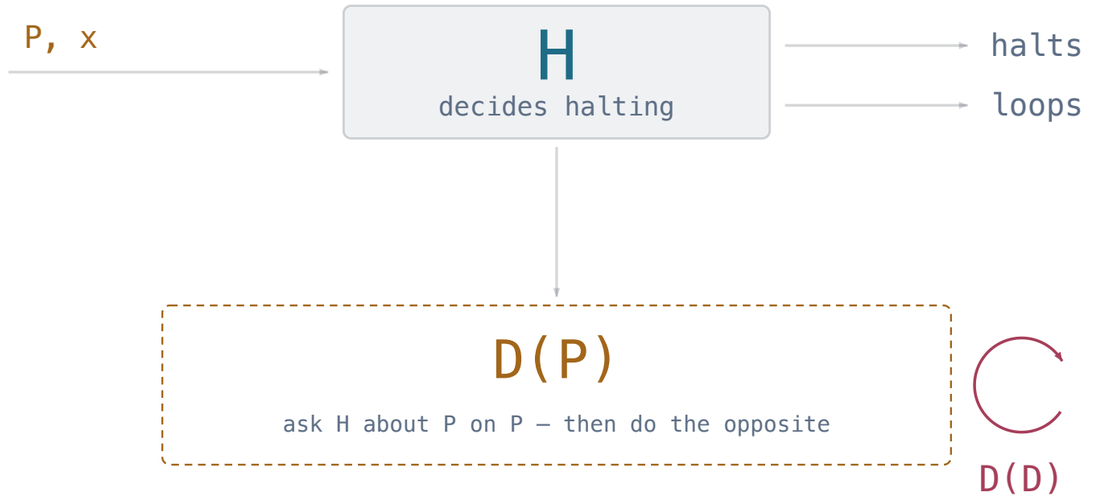
Run it: `./selfie -c loop.c -m 1`. It has not stopped yet. Will it?

You can reason: c is odd, adding 2 keeps it odd, and 0 is even — but the machine wraps at 2^{64}, so does c ever become 0? Odd plus even is odd, always, and wrapping preserves parity. It never stops.

You proved that. Could the compiler have? For this program, with effort, yes. For every program? That is today's question.

UNDECIDABILITY

Will this program ever stop? No program can always tell.



Suppose `H(P, x)` decides, for every program and input, whether P halts on x.

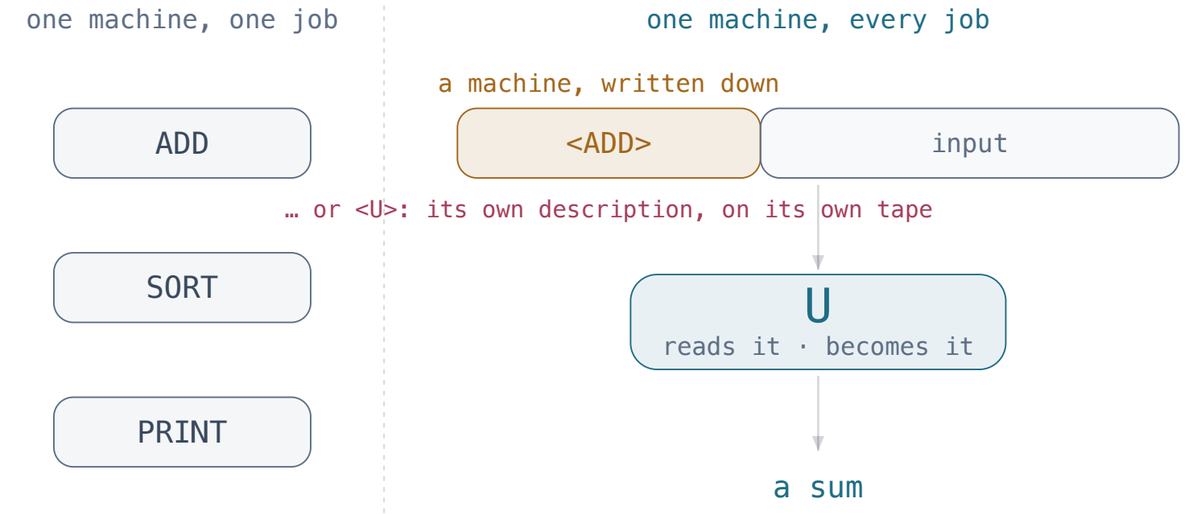
Build `D(P)`: ask H whether P halts on P — then do the opposite. A call, a conditional, a loop, a return. All in C*.

Now run `D(D)`. It halts exactly if it doesn't. Contradiction. So H never existed.

Turing, 1936 — the same paper that defined the universal machine, and thus invented the computer. The limit and the machine arrived together.

THE SAME MOVE, FORWARDS

One machine that can be any machine.



the job moved from the wiring to the tape

A machine used to *be* its job — to sort instead of add you built a different machine. Turing's move: put the machine's description on the tape, as data. Then one machine U reads any description and does whatever that machine would do.

That is universality: one piece of hardware, every possible behaviour, because the behaviour arrives as notation. Your phone is not phone-shaped. It is U holding a description.

Hand a decider its own description: contradiction, no H. Hand a machine any description: every program at once, one U. One sentence, read twice.

THE GENERAL CASE

It is not just halting. It is every interesting question about meaning.

RICE · 1953

> *Every non-trivial property of the behaviour of programs is undecidable. Only questions about their text are safe.*

DECIDABLE · ABOUT SYNTAX

How long is the code? Does it parse? Does it use this library? Are the types consistent? Does it contain a loop?

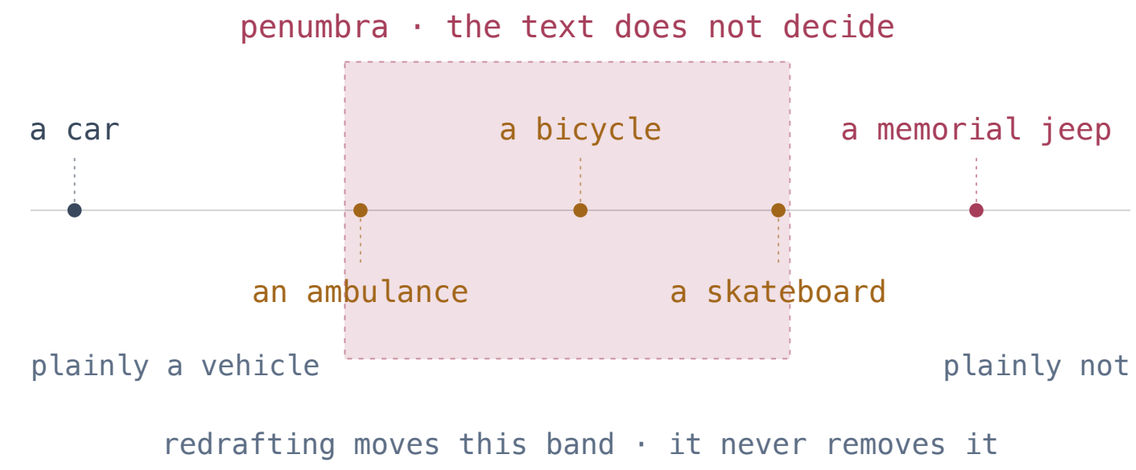
UNDECIDABLE · ABOUT SEMANTICS

Is it correct? Is it equivalent to that one? Does it ever divide by zero? Is it free of infinite loops? Is it safe?

So every practical tool — a type checker, a test suite, a linter, a model checker, a fuzzer, a proof assistant — is a deliberate approximation: sound but incomplete, or complete but unsound, or exact only within a bound. Choosing which to give up is the discipline.

INTERLUDE · LAW

No text contains its own application.



Hart's core, and Hart's penumbra

A statute says *no vehicles in the park*. A car, plainly. And then the arguing starts: an ambulance, a bicycle, a memorial jeep. No redrafting removes the penumbra — only moves it. The words are finite; the situations are not.

So law stops trying to settle meaning in the text and builds an institution: courts, appeals, precedent. Not a workaround — the only available design. Rice's theorem, with no mathematics in it.

PATTERN

One idea. Sixty years. Six theorems.

YEAR	WHO	THE LIST	THE DIAGONAL OBJECT
1891	Cantor	all real numbers	a number on no row
1901	Russell	all sets	the set of non-self-members
1931	Gödel	all provable sentences	"I am unprovable"
1936	Tarski	all definable predicates	"I am false"
1936	Turing	all decidable questions	a program that defies its judge
1953	Rice	all semantic properties	all of the above, at once

Assume a complete list. Ask each item about *itself*. Answer the opposite.
Learn the move once and you own the century.

REFRAME · 3 OF 5

No final authority means no ceiling.

WHAT IT FORBIDS

A machine, a method, or a person that settles all questions. No complete rulebook, no self-certifying system, no automatic correctness.

WHAT IT OPENS

Mathematics is not a finished building but an open frontier; engineering is a craft rather than a lookup; and judgement — yours — never becomes redundant.

> *Beware of bugs in the above code; I have only proved it correct, not tried it.*
>
> DONALD E. KNUTH, 1977

BEFORE NEXT WEEK

Recommended exercises.

1. Read the Meaning chapter from *The Machine, and What It Cannot Decide* to the end.
2. Write, in C*, the wrapper D for a hypothetical H. Then explain in one paragraph why no change to mipster could make it print "this program never halts" for all such programs.
3. Sort into text and behaviour: uses the keyword while; ever executes the while loop; contains the character /; ever divides by zero; is longer than 100 lines; prints the same as another program. Which can starc decide?
4. Name one approximation for each of: is it free of infinite loops; is it safe; is it correct. Say what each gives up.
5. Find the penumbra in a rule you live under — a house rule, a traffic rule, a grading rule.

NEXT WEEK

Systems: two ways to build an operating system, why they are the same, why virtualization is used anyway, and the loop at its centre — the kernel that needs what it provides.